Greedy algorithms/dynamic programming problem collection

http://www.cs.rochester.edu/~stefanko/Teaching/18CS282

1 Dynamic Programming

1.1 Subsequence Problems

0.1 Let $X = x_1, \dots, x_m$, $Y = y_1, \dots, y_n$ be two sequences. Our goal is to compute the length of the longest common subsequence[1] of X and Y. We will have a 2-dimensional table $T[0..m, 0..n]$, where

$$T[i,j] = \text{the length of the longest common subsequence of } x_1, \dots, x_i \text{ and } y_1, \dots, y_j.$$

Give an expression (or a piece of code) to compute $T[i,j]$ from the previously computed values of T (of course you will also need the values in X and Y in your expression).

0.2 Let $\overline{x} = x_1, \dots, x_m$, $\overline{y} = y_1, \dots, y_n$, and $\overline{z} = z_1, \dots, z_\ell$ be three sequences. Our goal is to compute the length of the longest common subsequence of $\overline{x}, \overline{y}$, and $\overline{z}$. We will have a 3-dimensional table $T[0..m, 0...n, 0..\ell]$, where $T[i,j,k]$ will be the length of the longest common subsequence of $x_1, \dots, x_i$, $y_1, \dots, y_j$, and $z_1, \dots, z_k$. Give an expression (or a piece of code) to compute $T[i,j,k]$ from previously computed values in T.

0.3 Given a string $\overline{x} = x_1 x_2 \cdots x_n$ we would like to find the length of the longest palindromic subsequence of $\overline{x}$ (a sequence is palindromic if it is the same as its reverse). Let $T[1..n, 1..n]$ be an array where $T[i,j]$ is the length of the longest palindromic subsequence of $x_i, \dots, x_j$ (note that, $T[i,j]$ is undefined for $j < i$). Give an expression (or a piece of code) for $T[i,j]$ in terms of already computed values in T.

0.4 Let $a_1, \dots, a_n$ and $b_1, \dots, b_m$ be two sequences of numbers. We would like to find **the longest common increasing subsequence** of $a_1, \dots, a_n$ and $b_1, \dots, b_m$. Give a *dynamic programming algorithm* for the problem. Clearly and succinctly describe: 1) the table are you using, 2) how do you recover the final answer from the table, and 3) the update rule.

0.5 We will call a sequence $b_1, \dots, b_m$ **barely changing** if neighboring numbers differ by at most 73 (that is, $(\forall i \in \{1, \dots, m-1\}) |b_i - b_{i+1}| \leq 73$).
Consider the following problem:
INPUT: A sequence $a_1, \dots, a_n$.
OUTPUT: The length of the longest barely changing subsequence of $a_1, \dots, a_n$.

We would like to solve the problem using dynamic programming. Let

$$P[i] = \text{the length of the longest barely changing subsequence of } a_1, \dots, a_i \text{ that ends with } a_i.$$

Give an expression (or a piece of code) that gives an efficient way of computing $P[i]$ (assume $i \geq 2$).

0.6 Let $a_1, \dots, a_n$ be a sequence of numbers. We want to find the increasing subsequence of $a_1, \dots, a_n$ with the largest sum. (For example if the input is $11, 1, 2, 3, 4, 12$ then the output is $11, 12$, a subsequence with sum 23.) We will compute a table $T[0...n]$ where $T[i]$ is the maximum sum of an increasing subsequence ending with a_i. Give an expression (or a piece of code) for $T[i]$ in terms of $a_1, \dots, a_i$ and $T[0], T[1], \dots, T[i-1]$.

[1]that is the largest ℓ such that there exist $1 \leq i_1 < \cdots < i_\ell \leq m$ and $1 \leq j_1 < \cdots < j_\ell \leq n$ such that $x_{i_k} = y_{j_k}$ (for all $k \in \{1, \dots, \ell\}$).

0.7 A sequence of numbers $b_1, \dots, b_k$ is convex if $2b_i \leq b_{i-1} + b_{i+1}$ for all $i \in \{2, \dots, k-1\}$. Given a sequence of numbers $a_1, \dots, a_n$ we want to find the longest convex subsequence of $a_1, \dots, a_n$. Describe the table (that is, the subproblems) and the update rule (that is, how you solve a subproblem using smaller subproblems).

0.8 We are given a sequence A of n numbers $a_1, \dots, a_n$. We would like to count the number of different subsequences of A. For example, $A = 1, 2, 3$ has 8 different subsequences (one of length 3, three of length 2, three of length 1, and one of length 0); on the other hand $A = 1, 1, 1$ has 4 different subsequences (one of length 3, one of length 2, one of length 1, and one of length 0). Give a dynamic-programming algorithm for the problem. Clearly describe the array you are using and how you compute the array.

0.9 We are given a sequence of n real numbers $A[1], \dots, A[n]$. We would like to select the longest subsequence such that any of its three consecutive terms sum to a positive number (zero is NOT a positive number). (In case you forgot what a subsequence is: a subsequence of length ℓ is $A[i_1], A[i_2], \dots, A[i_\ell]$ where the indices are increasing, that is, $1 \leq i_1 < i_2 < \cdots < i_\ell \leq n$).

For example, if the input is $10, -5, 2, -1, 0, 2, 0, -3, 1, 5$ then the longest subsequence satisfying the condition is $10, 2, -1, 0, 2, 0, 1, 5$.

We are going to solve the problem using dynamic programming. We are going to compute a table $T[1..n, 1..n]$ where $T[i, j]$ for $i < j$ will be the length of the longest subsequence satisfying the condition and ending with $A[i], A[j]$ (that is, the last two terms are $A[i], A[j]$). Give an expression to compute $T[i, j]$ from previously computed values.

0.10 "Common subsequence" We are given two sequences of real numbers $a_1, \dots, a_n$ and $b_1, \dots, b_m$. We want to find the common subsequence[2] of the two sequences that has the maximum sum.

EXAMPLE 1: For $1, 4, 2, 5, -11$ and $1, 2, 4, 5, -11$ the solution is $1, 4, 5$ (it has sum 10).

EXAMPLE 2: For $100, 1, 2, 3$ and $1, 2, 3, 100$ the solution is 100 (it has sum 100).

We want to solve the problem using dynamic programming. Let $T[i, j]$ be the maximum sum of a common subsequence of of $a_1, \dots, a_i$ and $b_1, \dots, b_j$. Give an expression (or a piece of pseudocode) to compute $T[i, j]$ from $T[i-1, j-1]$, $T[i, j-1]$, $T[i-1, j]$, and some of the $a_1, \dots, a_i$ and $b_1, \dots, b_j$.

1.2 String Problems

0.11 A **shuffle** of two strings A, B is formed by interspersing the characters into a new string, keeping the characters of A and B in the same order (for example, '**s**ev**e**r**al**' is a shuffle of 'seal' and 'evr'). Given three strings $A = a_1 \dots a_n$, $B = b_1 \dots b_m$, and $C = c_1 \dots c_{m+n}$, we would like to check whether C is a shuffle of A and B.

Consider the following proposed algorithm for to the problem. We compute a table $T[0..n, 0..m]$ using the following rules

- (for all $i \in \{0, \dots, n\}$) $T[i, 0]$ is true if and only if $a_1 \dots a_i = c_1 \dots c_i$,
- (for all $j \in \{0, \dots, m\}$) $T[0, j]$ is true if and only if $b_1 \dots b_j = c_1 \dots c_j$, and
- for other i, j the entry $T[i, j]$ is computed using the following update

$$T[i, j] = (c_{i+j} = a_i \text{ or } c_{i+j} = b_j) \text{ and } (T[i-1, j] \text{ or } T[i, j-1]).$$

[2] A subsequence of $a_1, a_2, \dots, a_n$ is any $a_{i_1}, a_{i_2}, \dots, a_{i_k}$, where $1 \leq i_1 < i_2 < \cdots < i_k \leq n$. Thus, e. g., $1, 3, 5$ is a subsequence of $1, 2, 3, 4, 5$.

Give an example of two stings A, B for which the algorithm produces wrong answer. Compute all the entries in the table T for your example (suggestion: keep your example small—that will save you work filing the table T).

0.12 A **shuffle** of two strings A, B is formed by interspersing the characters into a new string, keeping the characters of A and B in the same order (for example, '**sev**e**ral**' is a shuffle of 'seal' and 'evr'). Given three strings $A = a_1 \dots a_n$, $B = b_1 \dots b_m$, and $C = c_1 \dots c_{m+n}$, we would like to verify whether C is a shuffle of A and B. We are going to solve the problem using dynamic programming. We will compute a table $T[1..n, 1..m]$, where

$$T[i,j] = \begin{cases} \text{true} & \text{if } c_1 \dots c_{i+j} \text{ is a shuffle of } a_1 \dots a_i \text{ and } b_1 \dots b_j, \\ \text{false} & \text{otherwise.} \end{cases}$$

Give an expression (or a piece of code) for $T[i, j]$ (in terms of previously computed values of T and elements of A, B, C).

0.13 Let $X = x_1, \dots, x_m$, $Y = y_1, \dots, y_n$ be two sequences. Our goal is to compute the length of the longest common substring[3] of X and Y. We will have a 2-dimensional table $T[0..m, 0..n]$, where

$$\begin{gathered} T[i,j] = \text{the length of the longest common substring of } x_1, \dots, x_i \\ \text{and } y_1, \dots, y_j \text{ which ends with } x_i \text{ and } y_j \text{ (in the notation} \\ \text{from the footnote } s + \ell - 1 = i \text{ and } t + \ell - 1 = j). \end{gathered}$$

Give an expression (or a piece of code) to compute $T[i, j]$ from the previously computed values of T (of course you will also need the values in X and Y in your expression).

1.3 Knapsack problems

0.14 In the Knapsack Problem we have n items. The weight of the i-th item is $W[i]$ and the value of the i-th item is $V[i]$. Assume that the $V[i]$ are integers and $W[i]$ are real numbers. Let C be the capacity of the knapsack, and let M be the sum of the $V[i]$, that is, $M = \sum_{i=1}^{n} V[i]$. We would like to find a subset of items $S \subseteq \{1, \dots, n\}$ with total weight at most C and maximal value.

We will compute an array $K[0..M, 0..n]$, where entry $K[x, i]$ will be the minimal weight of a subset of $\{1, \dots, i\}$ with total value $\underline{\text{equal}}$ to x (if no subset of $\{1, \dots, i\}$ has total value x then the entry will be ∞). Give an expression (or a piece of code, if you wish) for $K[v, i]$ in terms of some of the $K[?, i-1]$ (the question mark should be replaced by appropriate expressions).

0.15 In the Knapsack Problem we have n items. The weight of the i-th item is Q_i and the value of the i-th item is Z_i. Assume that the Q_i's are small integers. Let C be the capacity of the knapsack (assume that C is an integer). We will compute an array $K[0..C, 0..n]$, where

$$K[B, i] = \text{the maximal total value of a subset of items } \{1, \dots, i\} \text{ with total weight at most } B.$$

Give an expression (or a piece of code) for $K[B, i]$ in terms of some of the $K[?, i-1]$ (the question mark should be replaced by appropriate expressions).

[3]that is the largest ℓ such that there exist $s \in \{1, \dots, m+1-\ell\}$ and $t \in \{1, \dots, n+1-\ell\}$ such that $x_{s+k} = y_{t+k}$ (for all $k \in \{0, \dots, \ell-1\}$).

0.16 In the KNAPSACK PROBLEM we have n items. The weight of the i-th item is $W[i]$ and the value of the i-th item is $V[i]$. Assume that the $V[i]$'s are integers and the $W[i]$'s are real numbers. Let C be the weight-capacity of the knapsack, and let M be the sum of the $V[i]$, that is, $M = \sum_{i=1}^{n} V[i]$. We would like to find a subset of the items with the maximal value where the total weight of the subset can be at most C.

We will compute an array $K[0..M, 0..n]$, where entry $K[x, i]$ will be the minimal weight of a subset of items $1, 2, \ldots, i$ with total value $\underline{\text{greater than or equal to}}$ x. Give an expression (or a piece of code) for $K[x, i]$ in terms of some of the $K[?, i-1]$ (the question mark should be replaced by the appropriate expressions).

0.17 In the KNAPSACK PROBLEM REVISITED I we have n items. The weight of the i-th item is $W[i]$, the value of the i-th item is $V[i]$, and the volume of the i-th item is $B[i]$. Assume that the $W[i]$'s and $B[i]$'s are integers and $V[i]$ are real numbers. Let C be the weight-capacity of the knapsack, and let D be the volume-capacity of the knapsack. We would like to find a subset of the items with maximal value where the total weight of the subset can be at most C and the total volume of the subset can be at most D.

We will compute an array $K[0..C, 0..D, 0..n]$, where entry $K[x, y, i]$ will be the maximal value of a subset of items $1, 2, \ldots, i$ with total weight at most x and total volume at most y. Give an expression (or a piece of code) for $K[x, y, i]$ in terms of some of the $K[?, ?, i-1]$ (the question marks should be replaced by the appropriate expressions).

0.18 In the KNAPSACK PROBLEM REVISITED II we have n items. The weight of the i-th item is $W[i]$ and the value of the i-th item is $V[i]$. Now we assume that the $W[i]$ are integers and $V[i]$ are real numbers. In contrast to the original knapsack problem we will assume that **the thief has 2 knapsacks** with capacities C_1 and C_2. We would like to find two disjoint subsets of items $S_1, S_2 \subseteq \{1, \ldots, n\}$ with maximal total value, and such that the total weight of S_1 is at most C_1 and the total weight of S_2 is at most C_2.

We will compute an array $K[0..C_1, 0...C_2, 0..n]$, where entry $K[x_1, x_2, i]$ will be the value of the solution restricted to items $\{1, \ldots, i\}$ with two knapsacks with capacities x_1 and x_2, formally,

$$K[x, y, i] = \max\left\{ \sum_{i \in S_1 \cup S_2} V[i] \;\middle|\; S_1, S_2 \subseteq \{1, \ldots, i\}, S_1 \cap S_2 = \emptyset, (\forall j \in \{1, 2\}) \sum_{i \in S_j} W[i] \le x_j \right\}.$$

Give an expression (or a piece of code) for $K[x_1, x_2, i]$ in terms of $K[x_1, x_2, i-1]$, $K[x_1 - ?, x_2, i-1]$, $K[x_1, x_2 - ?, i-1]$, $V[i]$ and $W[i]$.

0.19 In the KNAPSACK PROBLEM WITH UNLIMITED SUPPLY we have n types of items. The weight of the i-th type of item is $W[i]$ and the value of the i-th type of item is $V[i]$. Assume that the $V[i]$'s are integers and the $W[i]$'s are real numbers. Let C be the weight-capacity of the knapsack, and let M be the sum of the $V[i]$, that is, $M = \sum_{i=1}^{n} V[i]$. We would like to find a selection of the items with the maximal value where the total weight of the subset can be at most C. We have unlimited supply of each item.

We will compute an array $K[0..M, 0..n]$, where entry $K[x, i]$ will be the minimal weight of a selection of items of types $1, 2, \ldots, i$ with total value $\underline{\text{greater than or equal to}}$ x. Give an expression (or a piece of code) for $K[x, i]$ in terms of some of the $K[?, i-1]$ (the question mark should be replaced by the appropriate expressions).

0.20 We have n items. The weight of the i-th item is $W[i]$, the value of the i-th item is $V[i]$, and the volume of the i-th item is $B[i]$. Assume that $W[i]$'s and $B[i]$'s are integers and $V[i]$'s are real numbers. We have two robbers:

- "Weak" - with weight limit C and unlimited volume capacity,
- "Small" - with volume limit D and unlimited weight capacity.

That is, "Weak" can carry a subset of the items with total weight at most C and "Small" can carry a subset of the items with total volume at most D. There is no limit on the total weight of items "Small" can carry and there is no limit on the total volume of items "Weak" can carry.

The robbers colluded and want to steal a subset of the items with the maximum possible total value[4].

We will solve the problem using dynamic programming. Let $K[x, y, k]$ be the solution of the subproblem restricted to the first k items where "Weak" has weight limit x and "Small" has volume limit y. Give a formula (or a piece of pseudo-code) for $K[x, y, k]$ in terms of $K[\cdot, \cdot, k-1]$ (where $\cdot$'s are to be replaced by appropriate expressions). Clearly describe what kinds of optimal solutions do different parts of your expression address. (Do NOT worry about printing the final solution.)

1.4 Splitting Sequences

0.21 Let $a_1, \ldots, a_n$ be a sequence of integers. We would like to split the sequence into segments such that for each segment the first and the last number are the same. For example, sequence $1, 3, 2, 1, 4, 1, 5, 6, 7, 8, 4, 8, 9$ can be split into 4 segments: $1, 3, 2, 1$ and $4, 1, 5, 6, 7, 8, 4$ and 8 and 9; or into 6 segments $1, 3, 2, 1, 4, 1$ and 5 and 6 and 7 and $8, 4, 8$ and 9 (there are other ways of splitting).

We want to find the splitting that has the minimum number of segments (thus for the example above the first splitting into 4 segments is preferred). We are going to solve the problem using dynamic programming. We will compute a table $T[0...n]$ where $T[i]$ is the minimum number of segments to split $a_1, \ldots, a_i$. Give an expression (or a piece of code) for $T[i]$ in terms of $a_1, \ldots, a_i$ and $T[0], T[1], \ldots, T[i-1]$.

0.22 We define the **width** of a sequence S of numbers $b_1, \ldots, b_m$ as the difference between the largest and the smallest element in the sequence, formally

$$w(S) := \left(\max_{i\in[m]} b_i\right) - \left(\min_{i\in[m]} b_i\right).$$

For example $w(1, 2, 3, 4) = 3$ and $w(1, 1, 1) = 0$.

We have a sequence of n numbers $a_1, \ldots, a_n$. We want to split it into k sequences $S_1, S_2, \ldots, S_k$ where

$$S_1 = a_1, \ldots, a_{i_1}, \quad S_2 = a_{i_1+1}, \ldots, a_{i_2}, \quad S_3 = a_{i_2+1}, ..., a_{i_3}, \quad \ldots \quad S_k = a_{i_{k-1}+1}, \ldots, a_n.$$

The objective is to find a split (that is, values $i_1, \ldots, i_{k-1}$) that minimizes $w(S_1) + w(S_2) + \cdots + w(S_k)$ (that is, the sum of the widths of the resulting sequences). The input of the problem is n, k, and $a_1, \ldots, a_n$. We will solve the problem using dynamic programming. Let $T[i, \ell]$ be the value of the optimal solution of the problem for input i, ℓ, and $a_1, \ldots, a_i$ (that is, the minimal sum of widths of a split of $a_1, \ldots, a_i$ into ℓ sequences).

Write a **piece of pseudo-code** that, in $O(i)$ time, computes $T[i, \ell]$ using $T[1, \ell-1]$,...,$T[i-1, \ell-1]$ and $a_1, \ldots, a_i$.

[4]That is, we are maximizing $\sum_{i\in S} V[i] + \sum_{i\in T} V[i]$ over $S, T \subseteq \{1, \ldots, n\}$ such that $S \cap T = \emptyset$ and $\sum_{i\in S} W[i] \le C$ and $\sum_{i\in T} B[i] \le D$. Here S is the set of items given to "Weak" and T is the set of items given to "Small".

1.5 Merging Sequences

0.23 The **jumpiness** of a sequence of real numbers $c_1, \ldots, c_k$ is

$$\sum_{i=1}^{k-1} |c_{i+1} - c_i|.$$

For example the jumpiness of $1, 5, 2, 3$ is $8 = 4 + 3 + 1$.

A **shuffle** of two sequences $A = a_1, \ldots, a_n$ and $B = b_1, \ldots, b_m$ is formed by interspersing the numbers into a new sequence, keeping the numbers of A and B in the same order (for example, $2, 1, 100, 3, 101, 102$ is a shuffle of $2, 1, 3$ and $100, 101, 102$). Given two sequences $A = a_1, \ldots, a_n$ and $B = b_1, \ldots, b_m$ we want to find the minimal jumpiness of a shuffle of A and B. For example for $A = 1, 100$ and $B = 2, 101$ we should take $1, 2, 100, 101$ (jumpiness $= 100$).

We will solve the problem using dynamic programming. Let $\alpha[i, j]$ be the minimal jumpiness of a shuffle of $a_1, \ldots, a_i$ and $b_1, \ldots, b_j$ where the shuffle is required to end with a_i. Let $\beta[i, j]$ be the minimal jumpiness of a shuffle of $a_1, \ldots, a_i$ and $b_1, \ldots, b_j$ where the shuffle is required to end with b_j.

Give an expressions for $\alpha[i, j]$ and $\beta[i, j]$ using previously computed values ($\alpha[i-1, j], \beta[i-1, j], \alpha[i, j-1], \beta[i, j-1]$) and some of the values of $a_1, \ldots, a_i$ and $b_1, \ldots, b_j$.

1.6 Selecting subset with local constraints

0.24 We are given n positive numbers $a_1, \ldots, a_n$. A selection S of the numbers is called **valid** if no two consecutive numbers are selected in S. The goal is to find a valid selection of the numbers with the maximal sum. We are going to give an $O(n)$-time dynamic programming algorithm for the problem. We will have a table $P[1..n]$ where

$$P[i] = \text{the maximal sum of a valid selection of the first } i \text{ numbers.}$$

Give an expression (or a piece of code) for $P[i]$ in terms of $P[i-2]$, $P[i-1]$, and a_i.

0.25 We are given n numbers $a_1, \ldots, a_n$ (they can be positive or negative). A selection S of the numbers is called **valid** if

$$\underline{\text{out of each three consecutive numbers at least one is selected}}.$$

The goal is to find a valid selection of the numbers with the $\underline{\text{minimum}}$ sum. We are going to give an $O(n)$-time dynamic programming algorithm for the problem. We will have a table $P[1..n]$ where

$$P[i] = \begin{array}{l}\text{the minimum sum of a valid selection } S \text{ of the first } i \text{ numbers,}\\ \text{with the constraint that } i \text{ is selected (that is } i \in S).\end{array}$$

Give an expression (or a piece of code) for $P[i]$ in terms of $P[i-3]$, $P[i-2]$, $P[i-1]$, and a_i.

0.26 We are given n positive numbers $a_1, \ldots, a_n$ and a number $k \in \{1, \ldots, n\}$. The goal is to select a subset S of the numbers with the maximal sum and such that 1) no three consecutive numbers are selected in S, AND 2) the size of S is k. We will compute a table $T[0..n, 0..k]$ where $T[x, y]$ is the maximum sum of a valid subset of $a_1, \ldots, a_x$ where the size of the subset is y. Give an expression (or a piece of code) for $T[x, y]$ in terms of previously computed values of T.

0.27 Let $a_1, \ldots, a_n$ be a sequence of numbers. We want to find the longest increasing subsequence of $a_1, \ldots, a_n$ subject to the constraint that no two consecutive numbers can be chosen. (For example if the input is $1, 10, 2, 4, 5, 11, 6$ then the output is $1, 2, 5, 6$, a subsequence of length 4; note that $1, 2, 4, 5, 6$ is not valid since $2, 4$ are consecutive in the original sequence (also $4, 5$ are consecutive in the original sequence).) We will compute a table $T[0...n]$ where $T[i]$ is the maximum length of a valid increasing subsequence ending with a_i (valid means that no two consecutive numbers were chosen). Give an expression (or a piece of code) for $T[i]$ in terms of $a_1, \ldots, a_i$ and $T[0], T[1], \ldots, T[i-1]$.

0.28 Let $A = a_1, a_2, \ldots, a_n$ be a sequence of real numbers (any real numbers, including negative numbers are allowed). A selection of the numbers is called **valid** if

- no three consecutive numbers are selected, AND
- from each three consecutive numbers at least one is selected.

We want to find a valid selection of $a_1, \ldots, a_n$ with the maximum sum. We will solve the problem using dynamic programming. (We will only find the value of the optimal solution—do not worry about printing it.) Let

- $T[k, 0, 0]$ be the maximum sum of a valid selection of $a_1, \ldots, a_k$, restricted to selections in which a_{k-1} and a_k are NOT chosen,
- $T[k, 1, 1]$ be the maximum sum of a valid selection of $a_1, \ldots, a_k$, restricted to selections in which a_{k-1} and a_k are chosen,
- $T[k, 0, 1]$ be the maximum sum of a valid selection of $a_1, \ldots, a_k$, restricted to selections in which a_{k-1} is NOT chosen and a_k is chosen,
- $T[k, 1, 0]$ be the maximum sum of a valid selection of $a_1, \ldots, a_k$, restricted to selections in which a_{k-1} is chosen and a_k are NOT chosen.

Give expressions for $T[k, 0, 0]$, $T[k, 1, 1]$, $T[k, 0, 1]$, $T[k, 1, 0]$ in terms of $T[k-1, \cdot, \cdot]$ and a_k (where $\cdot$'s are to be replaced by appropriate expressions). Do NOT use any other entries in T, besides $T[k-1, \cdot, \cdot]$ in your expression (in particular, do not use $T[k-2, \cdot, \cdot]$ in your expression for $T[k, \cdot, \cdot]$).

1.7 Misc Problems

0.29 Let $m \leq n$ be positive integers. Let $P[1..n]$ be an array of positive real numbers. We are given a chocolate bar consisting of $m \times n$ squares. We would like to split the bar into 1×1 squares. We are only allowed to split one piece of the chocolate at a time using a vertical or a horizontal break. We have to pay $P[\ell]$ for a break of length ℓ. We would like to compute the minimal cost of breaking the bar.

We want solve the problem using dynamic programming. Let $M[i, j]$ be the minimal cost of breaking an $i \times j$ chocolate bar. In the space below write an expression for $M[i, j]$ in terms of

$$M[1, j], M[2, j], \ldots, M[i-1, j],\ M[i, 1], M[i, 2], \ldots, M[i, j-1], \text{ and } P[i], P[j].$$

(Example: a 2×3 bar can be broken as follows (using 2 breaks of length 2 and 3 breaks of length 1):

$$2 \times 3 \to 2 \times 2, 2 \times 1 \to 2 \times 1, 2 \times 1, 2 \times 1 \to \ldots \to 6 \text{ copies of } 1 \times 1$$

It can also be broken as follows (using 1 break of length 3 and 4 breaks of length 1):

$$2 \times 3 \to 1 \times 3, 1 \times 3 \to \ldots \to 6 \text{ copies of } 1 \times 1.$$

in this case the answer would be the smaller of $2P[2]+3P[1]$ and $P[3]+4P[1]$ (say, if, $P[1]=1, P[2]=2, P[3]=10$ then the minimal cost is 7)).

0.30 We are given a collection of n intervals $I_1,\dots,I_n$. Interval $I_i=[a_i,b_i]$ is assigned weight w_i. Assume that the a_i and b_i are all distinct. We want to find a max-weight subset of disjoint intervals, i. e., we want to find $S\subseteq\{1,\dots,n\}$ such that

- for any distinct $j,k\in S$ the intervals I_j and I_k are disjoint, and
- the $\sum_{j\in S} w_j$ is maximized.

Give an $O(n^2)$-time dynamic programming algorithm for this problem (clearly state what the content of the table is and explain the update rule).

0.31 We are given an $n\times n$ array A of zeros and ones. We want to find the size of the largest contiguous all-ones square. We are going to give a dynamic-programming algorithm with running time $O(n^2)$.

We will compute an $n\times n$ array P, where $P[i,j]$ is the size of the largest contiguous all-ones square whose bottom-right corner is i,j. We have $P[i,j]=A[i,j]$ if $i=1$ or $j=1$. If $i>1$ and $j>1$ then we will compute $P[i,j]$ using the values of $P[i-1,j], P[i-1,j-1], P[i,j-1]$, and $A[i,j]$. Give an expression (or a piece of code) for $P[i,j]$.

0.32 We are given an $m\times n$ matrix A filled with numbers. A **monotone path** starts at the top-left square and in each step moves either down or right. (For example if $m=2$ and $n=3$ then there are 3 monotone paths that end at the bottom-right square: right-right-down (visiting squares $(1,1)$, $(1,2)$, $(1,3)$, $(2,3)$), right-down-right (visiting squares $(1,1)$,$(1,2)$,$(2,2)$,$(2,3)$), and down-right-right (visiting squares $(1,1)$,$(2,1)$,$(2,2)$,$(2,3)$).) The **value** of a monotone path P is the sum of the entries of A in the squares visited by P.

We want to find the largest value of a monotone path that ends at the bottom-right square. We are going to solve the problem using dynamic programming. Let

$$T[i,j] = \text{the largest value of a monotone path that ends at square } (i,j)$$

Give an expression (or a piece of code) that gives an efficient way of computing $T[i,j]$ (assume $i\geq 2$ and $j\geq 2$):

0.33 We are given an $m\times n$ matrix A filled with numbers. A **monotone path** starts on the top-left square and in each step moves either down or right. For example if $m=2$ and $n=3$ then there are 3 monotone paths that end on the bottom-right square: right-right-down (corresponding sequence of entries: $A_{1,1}, A_{1,2}, A_{1,3}, A_{2,3}$), right-down-right (corresponding sequence of entries: $A_{1,1}, A_{1,2}, A_{2,2}, A_{2,3}$), and down-right-right (corresponding sequence of entries: $A_{1,1}, A_{2,1}, A_{2,2}, A_{2,3}$).

We are also given a sequence of numbers $b_1,\dots,b_k$. We want to know whether there exists a monotone path in A starting on the top-left square and ending on the bottom-right square whose corresponding sequence of entries is a subsequence[5] of $b_1,\dots,b_k$. We will solve the problem using dynamic programming. Let $T[i,j]$ be the smallest $z\in\{1,\dots,k\}$ such that there exists a monotone path in A starting on the top-left square and ending on the (i,j)-square whose corresponding sequence of entries is a subsequence of $b_1,\dots,b_z$. If no such $z\in\{1,\dots,k\}$ exists then we set $T[i,j]=\infty$. Give an expression (or a piece of pseudo-code) that computes $T[i,j]$ (from previously computed values in T and the input).

[5] A **subsequence** of a sequence $b_1,\dots,b_k$ is a sequence $b_{i_1}, b_{i_2},\dots,b_{i_\ell}$ where $\ell\in\{0,\dots,k\}$ and $1\leq i_1<i_2<\cdots<i_\ell\leq k$.

0.34 We have a 23×23 chessboard filled with numbers (each square has one number). We want select a subset of the squares so that 1) their total sum is maximized, and 2) there is no 2×2 square which has all of its squares selected. Give an algorithm for the problem, argue why is it correct, and state the running time of your algorithm. Pseudocode is not required but you should give a very clear description of your algorithm. The running time of your algorithm should be such that when implemented on today's machines it will finish in less than a minute.

0.35 A folding meter stick consists of n pieces connected by hinges. The pieces have lengths $a_1, \dots, a_n \in \{1, \dots, M\}$. We would like to fold the meter into a segment of the shortest possible length. (For example, if $a_1 = 17, a_2 = 13, a_3 = 30$ then the answer is 30 (we bend the second hinge).)

Here is a formalization of the problem (there is no need to read it if you understand the description above):

INPUT: integers M, n, a sequence of integers $a_1, \dots, a_n \in \{1, \dots, M\}$
OUTPUT: the minimum of the following expression over ± 1 sequences $s_1, \dots, s_n$

$$\left(\max_{j \in \{0,\dots,n\}} \sum_{i=1}^{j} a_i s_i \right) - \left(\min_{j \in \{0,\dots,n\}} \sum_{i=1}^{j} a_i s_i \right).$$

Give an efficient dynamic programming algorithm for the problem. Clearly state what is the content of your array.

2 Greedy Algorithms

0.36 For each collection of coin values below determine whether the greedy algorithm always produces an optimal solution (YES = greedy always optimal, NO = greedy sometimes not optimal).

1. $1, 2, 5$ YES - NO

2. $1, 5, 11$ YES - NO

3. $1, 4, 13$ YES - NO

4. $1, 2, 4, 8, 16$ YES - NO

0.37 Recall the activity selection problem: An activity is a time interval [start-time, end-time]. Two activities are said to conflict if they overlap. The goal is to select the largest subset S of given activities such that no pair of (distinct) activities in S conflict. For each of the following 4 greedy algorithms decide if it always gives an optimal subset of the activities. Circle the correct answer (YES = always optimal, NO = sometimes not optimal).

1. Start with $S = \{\}$, $Z =$ INPUT. Repeat the following until no activity is left. Pick the activity $a \in Z$ with the <u>earliest start-time</u>, put a into S, and remove all activities in Z conflicting with a.

   YES - NO

2. Start with $S = \{\}$, $Z =$ INPUT. Repeat the following until no activity is left. Pick the activity $a \in Z$ with the earliest end-time, put a into S, and remove all activities in Z conflicting with a.

YES - NO

3. Start with $S = \{\}$, $Z =$ INPUT. Repeat the following until no activity is left. Pick the activity $a \in Z$ with the latest start-time, put a into S, and remove all activities in Z conflicting with a.

YES - NO

4. Start with $S = \{\}$, $Z =$ INPUT. Repeat the following until no activity is left. Pick the activity $a \in Z$ with the latest end-time, put a into S, and remove all activities in Z conflicting with a.

YES - NO

0.38 Find frequencies for symbols a, b, c, d such that the the following table gives Huffman encoding (put the frequencies in the third column).

a	0	
b	10	
c	110	
d	111	

0.39 Consider the coin-change problem with coin values $1, 2, 6, 13$. Give an amount for which the greedy algorithm does not use the minimal number of coins.